BG3 Custom Surface Creation Guide

A Guide by StorybittenSoul.

Surfaces have many interesting interactions in the game, from clouds being removed by Gust of Wind to any Surface being used for a creature's blood when injured or killed. With this guide, you too can create your own custom Surfaces.

Note that while it is possible to do this without the use of the Official Modders Toolkit, the Toolkit simplifies many of the steps and is the way I would recommend working on your Surface.

Custom Surface Mod Reference List

To help you avoid conflicts, this guide will list known mods that use the techniques you are about to learn.

- Bloodrage Mist Surface by StorybittenSoul uses SurfaceBloodCloud

Definitions

Surface: When you see “Surface” with a capital-S, this guide is referring to the thing we are trying to edit or create. If it is not capitalized, this means the guide is referring to some other kind of surface, such as the ground. Example- “Now to test your Surface by casting the spell on a nearby open surface.”

TK: This means the BG3 Toolkit.

HC: The guide will include HC (short for Hard-Coded) when referring to anything that cannot be changed with the TK or by any other method known at the time this guide was written.

Entity: A character, item, or other thing which has a RootTemplate. To avoid confusion, this guide will not include Surfaces as entities.

[ ]: Any time code syntax is being given, the use of brackets around text will indicate a placeholder that you should replace with the relevant name, number, or other bit of code. The text inside the brackets will give an indication as to the expected code it should be replaced with.

Status: In this guide, the word Status will usually refer to a Boost that has been added to a Surface in the Status Data field. This is because the Status may have additional behavior that is not defined in the Boost.

Step 1: Planning and Base Surface Selection

This step is vital to your Surface working. Since some Surface behaviors are HC, there is a limit to what is possible to make.

Technically speaking, creating a new Surface in BG3 can't be done. All Surfaces are HC into the game and even the Unlocked TK cannot add a new Surface. That having been said, there are a few Surfaces Larian coded into the game that did not end up being used, and they can be modified (to an extent) to create new ones! There are some limitations, especially regarding Ground-Layer Surfaces, so keep that in mind as you proceed.

Planning

First, you must decide what your Surface will do. Where would it occur in the game? Is it only accessible as part of a specific spell, or could it appear from a variety of magical or mundane sources? What will it look like? What effects would it have? Most importantly, how do other effects and Surfaces interact with it? This last question will be where you encounter the most obstacles, as these interactions are HC.

Base Surface Selection

Take a look at the following chart and see if one of the listed Surfaces is compatible with your idea.

ENUM	SURFACE	Name in Game	Layer	HAS VISUALS ?	USED IN GAME?	USED IN MOD?	Known Surface Change Interactions
5	BloodElectrified	Electrified Blood	Ground	YES	UNUSED		Deelectrifies to Blood. Doesn't freeze.
6	BloodFrozen	Frozen Blood	Ground	YES	UNUSED		Melts to Blood.
17	TrialFire	Fire	Ground	YES	UNUSED?		Douses to Ash. Can Ignite and Vaporize other Surfaces.
46	BloodCloud	Blood Cloud	Cloud	NO	UNUSED	Bloodrage Mist	None
57	PotionHealingGreaterCloud	Potion of Greater Healing	Cloud	YES	USED* - BREAK HEALING POTION		None
58	PotionHealingSuperiorCloud	Potion of Superior Healing	Cloud	YES	USED* - BREAK HEALING POTION		None
59	PotionHealingSupremeCloud	Potion of Supreme Healing	Cloud	YES	USED* - BREAK HEALING POTION		None
62	PotionVitalityCloud	Potion of Vitality	Cloud	YES	USED* - BREAK POTION OF VITALITY		None
63	PotionAntitoxinCloud	Antitoxin	Cloud	YES	UNUSED		None
64	PotionResistanceAcidCloud	Potion of Acid Resistance	Cloud	YES	USED* - BREAK ELIXIR OF ACID RESISTANCE		None
65	PotionResistanceColdCloud	Potion of Cold Resistance	Cloud	YES	USED* - BREAK ELIXIR OF COLD RESISTANCE		None
66	PotionResistanceFireCloud	Potion of Fire Resistance	Cloud	YES	USED* - BREAK ELIXIR OF FIRE RESISTANCE		None
67	PotionResistanceForceCloud	Potion of Force Resistance	Cloud	YES	UNUSED(ELIXIR NOT FOUND IN UNMODDED GAME)		None
68	PotionResistanceLightningCloud	Potion of Lightning Resistance	Cloud	YES	USED* - BREAK ELIXIR OF LIGHTNING RESISTANCE		None
69	PotionResistancePoisonCloud	Potion of Poison Resistance	Cloud	YES	USED* - BREAK ELIXIR OF POISON RESISTANCE		None
NONE	RockSpikes			NO	DOES NOT WORK		N/A
NONE	BlackPowderExplode			NO	DOES NOT WORK		N/A
NONE	BlackRain	Black Puddle		NO	DOES NOT WORK		N/A
NONE	ConfusionCloud			NO	DOES NOT WORK		N/A
NONE	DaggersCloud			NO	DOES NOT WORK		N/A
NONE	DaggersCloudElectrified			NO	DOES NOT WORK		N/A
NONE	DeepWaterRunning			NO	DOES NOT WORK		N/A
NONE	GreaseFrozen			NO	DOES NOT WORK		N/A
NONE	NarcolepticCloud			NO	DOES NOT WORK		N/A
NONE	DrowPoison			NO	DOES NOT WORK		N/A
NONE	PotionAnimalSpeakingCloud			NO	DOES NOT WORK		N/A
NONE	PotionRemedialCloud			NO	DOES NOT WORK		N/A
NONE	PoitionHillGiantStrengthCloud			NO	DOES NOT WORK		N/A

*These Surfaces are used in the game, but might go unmissed if repurposed. See SIDEBAR: Potion and Elixir Surfaces for more information.

As you can see, there are only 5 truly unused Surfaces in the game. Three of those are Ground-Based Surfaces, meaning they will interact with only things which touch the ground and cannot be blown away. The other two are Clouds, so they can be blown away and can affect anything that enters their area, even if they are floating. Furthermore, all of the Ground-Based ones available have very specific HC interactions, while the Cloud-Based Surfaces have none besides the HC reaction to wind- and cannot be given any.

So, unless your desired Surface is going to logically turn to blood when Melted/Delectrified, or ash when Doused, or is a cloud that would *not* have any reaction to Cold, Lightning, or Fire damage whatsoever, you are out of luck.

There is still a lot of excellent creative space to work with- perhaps you use FrozenBlood to create a Surface meant to be a specific custom creature's poisonous blood (which loses potency after several rounds or when "burned off" with fire damage). You might also envision using TrialFire to create a new rival of Hellfire or Holy Fire, associated with another Outer Plane or even inspired by mythical Greek Fire.

The abundance of Cloud-Layer Surfaces can open up even more options, such as areas of Wild Magic, Curses, Ectoplasm (actually that might make for a good use of Electrified Blood), Solidified Time, The Smell of Baked Cookies, or anything else you care to imagine, provided that you don't expect it to explode, or Condense/Electrify into another Surface.

Now that you've got the premise of your Surface outlined and you've chosen a base Surface, it's time to begin making your changes.

Step 2: Overriding the RootTemplate

If you're using the TK for this, you'll want to start it up now and open (or create) the Project that will include your custom Surface. Basic_Level_A is probably the best level for testing your Surface at this stage.

2.1 In the RootTemplates, locate the Base Surface you chose in Step 1. You can filter the list to only include Surfaces by Right-Clicking the Orange Flame Icon at the top of the RootTemplates pane.

SIDEBAR: Potion and Elixir Surfaces

Some elixirs and potions have custom surfaces and others do not. Specifically, the Resistance Elixirs listed in the above table all do, but the Elixirs of Necrotic, Psychic, Radiant, and Universal don't. Meanwhile, Force Resistance Elixirs have a Surface despite having been removed from the game.

This indicates that Resistance Elixir Surfaces, as well as Surfaces for most potions, were eventually deemed unnecessary by Larian and dropped (or outright removed) during development. This is reinforced by the fact that Larian created an Antitoxin Surface, but never assigned it to the breaking of an Antitoxin.

All of these could easily be repurposed without anyone missing anything, but the relevant potion or elixir would need to have its corresponding Surface removed from its thrown/breaking effects.

In a similar vein, the Potion of Vitality is rarely found in-game, and it is doubtful that many are using it by throwing/breaking it and running through the Surface, which makes it a promising choice if you wish to avoid conflicts with other custom Surfaces.

Lastly, the healing potions might be changed to use all the same Surface, as the visual distinction between them is minor enough, though this would make them all heal the same amount without some workaround.

If you are truly desperate, Mysterious Liquid is identical to Oil in all but name and appearance, Changing the Nautiloid Tanks to spill Oil would free up a Surface… provided you and your mod downloaders don't mind the visual change to it, and you are careful to account for EVERY instance where Mysterious Liquid might appear whether it is upon level load or from a broken Nautiloid Tank.

The following Surfaces are functionally the same (but again, be VERY careful and thorough if repurposing them):

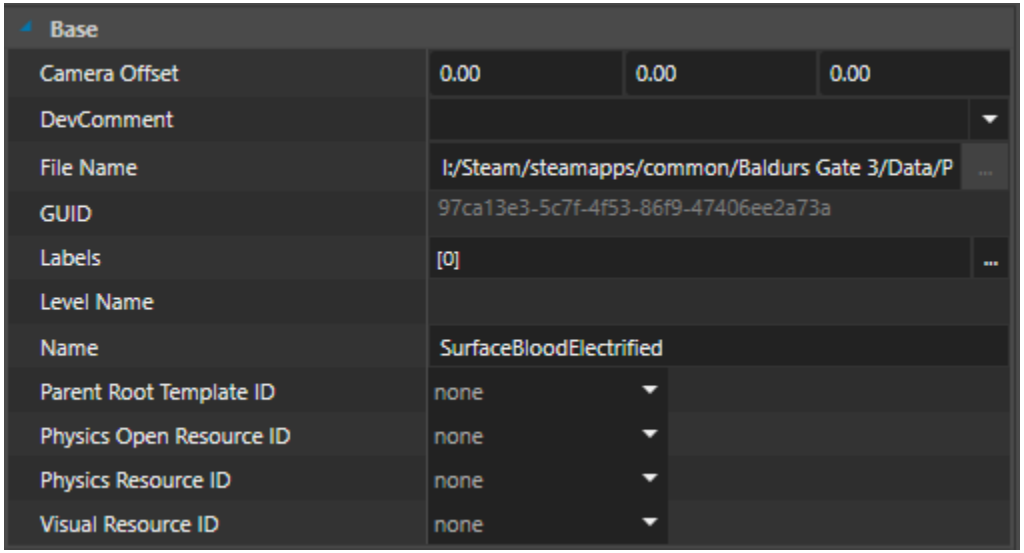
- SurfaceAlienOil and SurfaceOil
- SurfaceBloodSilver and SurfaceBlood
- SurfaceSewer and SurfaceGrease

2.2 Once you've found the Base Surface, Right Click on it and select "Override In Active Mod" to claim it. Do NOT change the Name, because the Surface's behavior is HC based on that name. All changes can be done in later steps, so it is okay to save the Surface as-is at the moment. Once you've saved it, the listing will have a green checkmark next to it. If at any point you decide you want to *permanently* revert all your changes to this Surface, you can delete it and restart the TK.

Great work, you can now begin making changes to the base and begin creating your custom Surface.

Step 3: Basic RootTemplate Adjustments

There is a lot of potential here, and it's not always easy to tell what each field does, so we'll take it one section at a time.

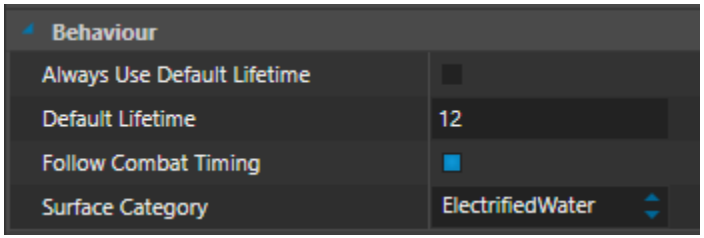


Section 1 - Base

Unlike most Root Templates, this Section is relatively unimportant to us. If your Surface is a variant of another compatible Surface, you may wish to set that to the appropriate Surface. It is not recommended to change any other fields here. Whatever you do, **DO NOT CHANGE THE NAME FIELD.** Your Surface might not end up having any relation to its Base Surface, but its connection to the Name is HC, so changing it will make a Surface that simply doesn't work until it is changed back. See Section 6 for details on changing the display name which players see.

Section 2 - Behavior

You may or may not need to change things in this section.



- The **Always Use Default Lifetime** checkbox, if checked, will do exactly that; No matter how the Surface is created, it will last for the Lifetime defined in the next field.

- **Default Lifetime** is how long the Surface will exist if no other duration is defined. For instance, when Water is turned to WaterFrozen via Cold damage, it will stay frozen for the Default Lifetime. If created via a spell such as Sleet Storm, WaterFrozen will use the duration outlined in the CreateSurface entry in Sleet Storm's Spell Properties instead of the default. This Lifetime is in seconds. (As a reminder, a Round is 6 Seconds).

- If the **Follow Combat Timing** box is NOT checked, the Surface's Lifetime will tick down even while in combat or turn-based mode. If it IS checked, then combat initiated near the Surface will cause it to only tick down at the end of a round until combat ends.

- The **Surface Category** setting does not change the Surface behavior per se, but rather how the Surface interacts with other Surfaces and some objects. All Surfaces tagged as Fire will try to Ignite, Melt, or Vaporize any other Surface it touches. Changing it to ElectrifiedWater will make it

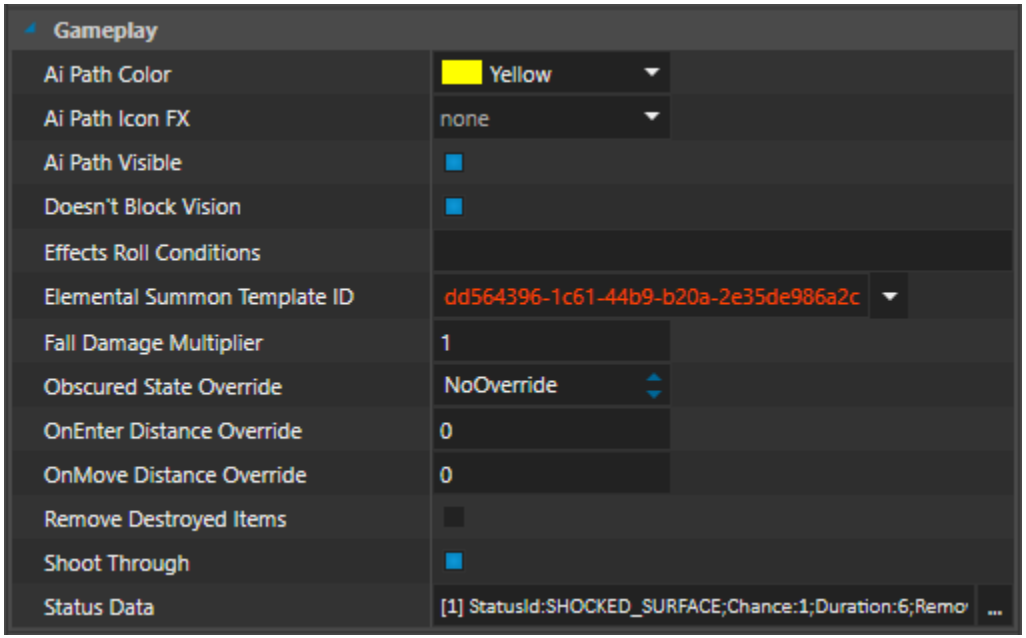
attempt to Electrify other Surfaces. These settings do not change any other aspect of the Surface's behavior, so setting a Surface Category to Water does not make it able to be Frozen, Electrified, or Vaporized.

Of important note are the settings Dominant, Inert, and Irreplaceable (and their Cloud equivalents).

- **Dominant** means that this Surface will replace any Surface it overlaps (on the same layer), but it can also be replaced by other Surfaces.
- **Inert** means it will NOT replace any Surface it overlaps, but other Surfaces can replace it.
- **Irreplaceable** will replace any Surface it overlaps and it will NOT be replaced by other Surfaces. Two Irreplaceable Surfaces that overlap will interact as if they were Dominant instead. This means that Chasms can be overwritten by Irreplaceable Ground-Layer Surfaces, so be careful!

Section 3 - Gameplay

This section is likely where most of your customization will be.



- **Ai Path Color** is the color a player's path trajectory will change to when they are being shown their potential movement path over or through this Surface. Typically (but not always), white means the entity's movement will not be affected by the Surface, Yellow means the Surface might reduce or halt the entity's movement, and Red means the Surface will deal damage or is otherwise very harmful to the entity.

- **Ai Path Icon FX** will provide a special Icon when a player hovers over this Surface while planning their movement path, right where the path ends. It will not replace the Cursor. Most Surfaces do not use this.

- **Ai Path Visible**, if NOT checked, will make the portion of the player's path that travels through it unable to be seen (making Path Color irrelevant). Path Icons will still appear if included.

- **Doesn't Block Vision** means that entities with vision can see through this Surface. Uncheck this for Surfaces similar to Darkness or Fog Cloud which are meant to blind characters or affect targeting entities within or outside of the Surface. On its own this will not prevent those outside the Surface from targeting those within or on the other side of the Surface, but combined with other settings that can be achieved. Some Wall/Surface combos may also want this unchecked.

- **Effects Roll Conditions** refers to the Saving Throw made by entities to avoid the Surface's effects (defined in Status Data below). If blank, no Save is granted to the entity.
  - If you do wish to include a Saving Throw, the syntax is `[OptionalCondition1()] [and/or] [OptionalCondition2()] [and] not SavingThrow(Ability.[AbilityScore],[DC],[OptionalModifiers])`.

    - An example from Grease is `Character() and not SavingThrow(Ability.Dexterity,SourceSpellDC(12),AdvantageOnSlipping())` which would mean that it only causes characters to fall prone (not objects), and only works on those that fail a Dexterity Saving Throw with a DC equal to the Spell DC of the caster (or DC 12 if not created by a spell), and gives Advantage to the Saving Throw if the character has a relevant ability which would provide that.
    - Adding a Condition like `Enemy()` without the `not SavingThrow()` would cause this Surface to only work on those that meet the condition without making them roll, in this case making it only work on enemies of the Surface's creator. SurfaceGithPheremoneGasCloud is a good example of this.

- **Elemental Summon Template ID** does nothing at this time. Its purpose would normally be to assign an entity (via its UUID) that could be summoned by the Surface under the right conditions. Unfortunately, it appears Larian did not fully implement this, as no developers or modders have found or shared an official way to reference this field. Use of Script Extender should be able to reference this field, but is outside the scope of this guide. This will be updated if new information comes to light.

- **Fall Damage Multiplier** will multiply damage received from an entity falling into this Surface. As an example, SurfaceWeb has a multiplier of 0, meaning that all fall damage is negated when landing on it. A setting of 0.5 would halve fall damage, 2 would double it, et cetera.

- **Obscured State Override** refers to whether the area is considered Obscured, Lightly Obscured, or Clear (as in well-lit) for the purposes of sneaking and spotting Stealthed characters. Setting this to NoOverride has no effect on Stealth, so the light level is used as the default. You can use this setting to improve the situation for those trying to hide, or for a Surface that creates light, set it to Clear to make it worse for Stealth. Even if this is set to Obscured and Doesn't Block Vision is unchecked above, a character can still be targeted unless they enter Stealth. Unlike the light level, this kind of Obscuring SHOULD stop those with Darkvision from seeing the character.

- **OnEnter Distance Override** is for effects set in Status Data that trigger upon entering the Surface. This is the distance, in meters, an entity can enter before such effects trigger. 0 means that stepping over the threshold counts as "entering" it, which is the most common setting. If the OnEnter Distance ends up being larger than an instance of this Surface, it will act as if the entity never entered it.

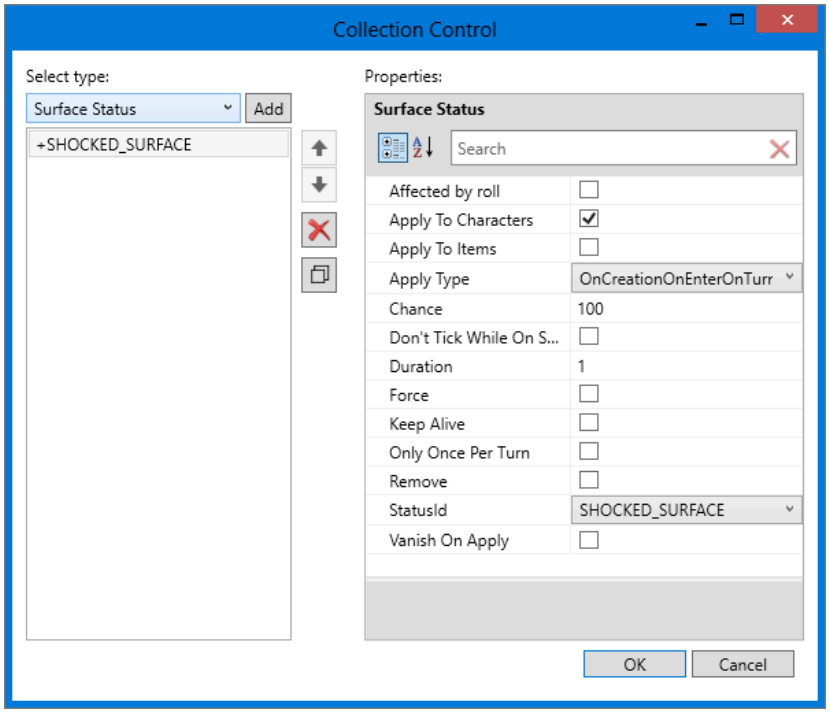
- **OnMove Distance Override** works similarly to the previous field. If an entity is already within the Surface and moves through it, this defines how far, in meters, it will get before the effects apply. In Status Data, effects can be set to apply every time the entity moves the set distance, or only the first time it does. For both of these overrides, Ground-Based Surfaces only affect entities touching the ground; If an entity takes no steps before jumping, being thrown, or flying out (providing it ends outside the Surface) it will not trigger these effects, even at 0 Distance. Cloud-Based Surfaces will still have an effect when entering or passing through them in any manner.

- **Remove Destroyed Items,** if checked, will cause any objects which are destroyed by or within the Surface to fade away afterward. Otherwise, their debris will remain. Most Surfaces will not have this checked.

- **Shoot Through** works like Doesn't Block Vision, except that projectiles such as arrows (and any Projectile spell, like Eldritch Blast) will treat this Surface as a barrier if this is NOT checked. Combine this with unchecking Doesn't Block Vision and setting Obscured State Override to Obscured to prevent targeting entities inside the Surface regardless of Stealth.

- Status Data has the bulk of the effects of the Surface defined within. These Boosts will be applied to any entity within the Surface. If you defined any conditions in Effects Roll Conditions, this is where you can decide which Statuses are applied by that roll. If you set a Status to trigger OnEnter or OnMove, the override fields above will affect those statuses. You will need to click the ellipses (...) button to the right of the field to open a window which facilitates adding or changing the effects that the Surface creates. See SIDEBAR: Adding and Changing Surface Status Data for how to use the window.

SIDEBAR: Adding and Changing Surface Status Data

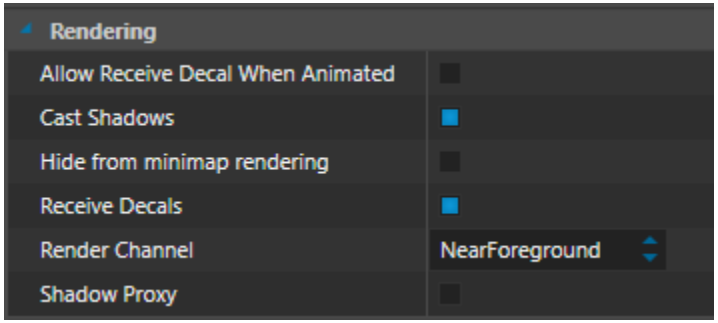


On the left side, press the Add button to create an entry which you can edit. It will be called "(none)" until a Boost is defined in the StatusID field.
You can change existing Statuses, or if you wish to delete one, select it and press the red X button. The arrows let you determine the order that Statuses are applied. Note that Copying Statuses does not always copy correctly.

With a Status entry selected, on the right side you can determine the status' properties.

- **Affected by roll** should be checked if this Status relies on a Saving Throw defined earlier.
- **Apply to Characters** should be checked if this Status can be applied to creatures.
- **Apply to Items** is the same as above but for objects and the like.
- **Apply Type** determines what actions or events from an entity cause the Status to try to be applied. There are many combinations, so choose carefully. OnCreation triggers the moment an instance of the Surface appears. If included, OnEnter and OnMove will take into account the Distance Overrides defined earlier (if any).
- **Chance** is a percentage in whole number form (1 to 100) that those within will be affected by this Status when the Apply Type is triggered. A setting of 50 would cause it to occur only half of the time. This does NOT take into account Saving Throws, so a setting of 25 would prompt Saving Throws only 1/4th of the time, and succeeding the Saving Throw would still negate the Status.
- **Don't Tick While On Surface** means that the Duration of the Status will not tick down while the entity is within the Surface's area.
- **Duration** is the time (in Rounds) that the Status will persist on the entity. Setting to 0 causes the Status to disappear immediately (unless Don't Tick is checked and the entity is still on the Surface). Setting to -1 makes the Status permanent.
- **Force** will make all affected entities that trigger the Apply Type experience this Status, even if they succeed on a Saving Throw or would normally have Immunity to this Status.
- **Keep Alive** means that the Status persists only while on the Surface, removing the Status if the entity leaves (even if Duration is -1).
- **Only Once Per Turn** will make the entity only trigger OnMove once per turn, even if it moves around repeatedly on the Surface. The Distance is defined earlier in OnMove Distance Override.
- **Remove** causes the defined Status to be removed from the entity if it already has the status. For instance, WaterCloud causes the Burning status to be removed from those inside the cloud's area. Statuses in the Status list will have the + turn to a - if this is checked, to quickly identify which is being added and which is being removed.
- **StatusID** is a dropdown where you select the text ID of the Boost being applied/removed. This refers to the StatusBoost entries in Stats. The dropdown is sorted alphabetically and can be slow to load. See the Boosts section later in this guide for more information.
- **Vanish On Apply** will make the entire Surface disappear once any applicable entity has received this Status from it. If a Surface has a default state (like WaterFrozen becomes Water when its duration expires) it will change into that default Surface instead of disappearing.

Section 4 - Rendering

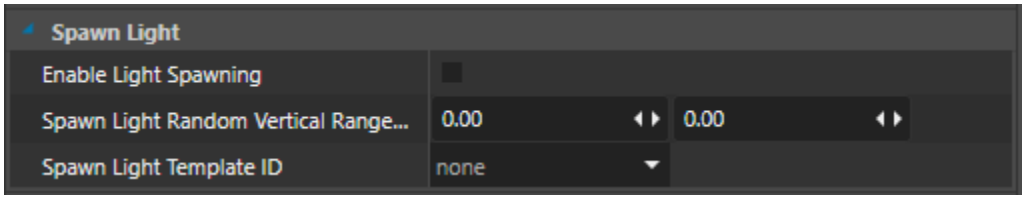


Most of these settings are the same for all Surfaces. If the Surface is made of darkness or shadows (Such as SurfaceDarknessCloud), uncheck **Cast Shadows**. If the Surface should NOT display a texture defined in Section 7 - Visual, uncheck **Receive Decals**. Note that only one default Surface in the game (SurfaceBladeBarrier) has this unchecked.
Allow Receive Decal When Animated, **Hide from minimap rendering**, and **Shadow Proxy** are normally unchecked. The normal **Render Channel** is NearForeground.

If you experiment with these settings and find useful behaviors, please inform the author of this guide and it will be updated.

Section 5 - Spawn Light

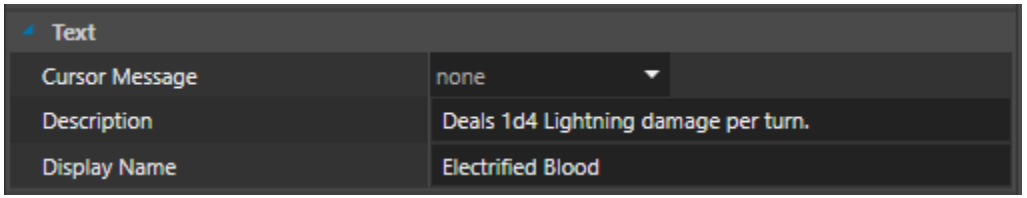
Here is where you'll input how much light the Surface emits. A supernatural liquid Surface like SurfaceAlienOil might have a slight glow. A Fire Surface likely gives off a flickering light. A Daylight Surface emits a bright, constant glow. Clouds generally do not emit light, but this could make an interesting effect.



Check the **Enable Light Spawning** box if this Surface should create any source of light.
The lights will spawn in random places across the Surface.
The **Spawn Light Random Vertical Range Offset** fields define the potential height above the Surface that the lights will appear, the left field being the lowest point and the right field being the highest. 0.00 is the Ground. The flames that appear as part of SurfaceFire spawn at random heights between 0.01 and 0.10. The mild glow from SurfaceAlienOil can spawn up to 0.20 above it. SurfaceHellfire's particularly large flames can spawn between 0.25 and 0.50 above the Surface.
Spawn Light Template ID is the name of the Root Template that defines the color, shape, brightness, and other aspects of the lights. You should be able to browse through these by right-clicking the yellow light bulb icon of the RootTemplates selector pane. Creating new Light Templates is outside the scope of this guide.

Section 6 - Text

Your Surface's basic description and other text is edited here.

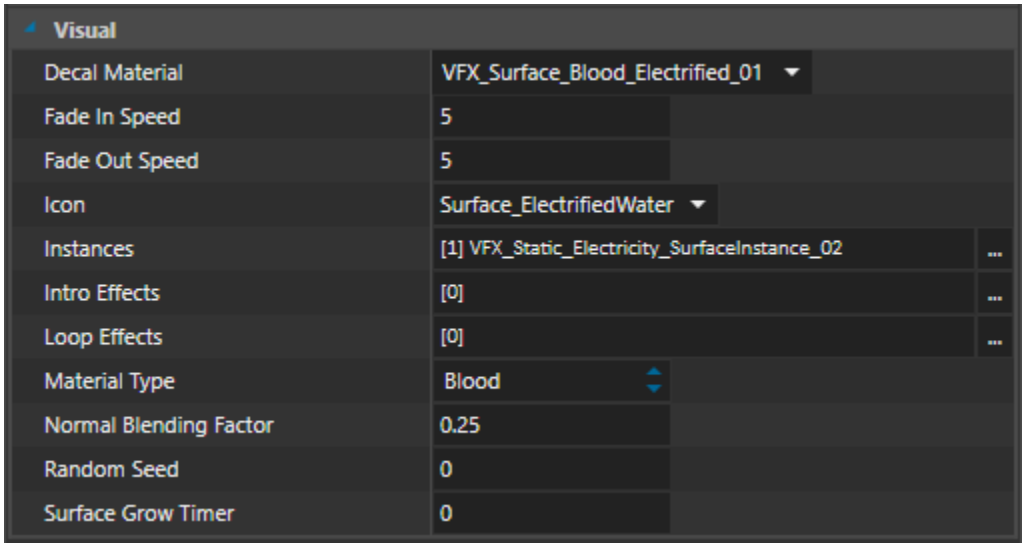


Cursor Message is a bit of text that appears when the player's cursor hovers over the Surface. None of the game Surfaces use this feature, but if you want to include one, the message needs to be defined in the UUID Object Editor , under the Surface section.

Description is what will display when Examining a Surface (On PC this is usually by hovering over it and pressing the T key), or when it is mentioned in an ability tooltip. It is best to keep this short and helpful.

Display Name is what players will see when they hover over the Surface, as well as the name displayed in any Tooltips. This has no bearing on the coded aspects of the Surface, so set it to whatever you'd like the Surface to be called.

Section 7 - Visual



Besides lighting and shadows, all of the Surface's visual effects (VFX) are determined in this section.

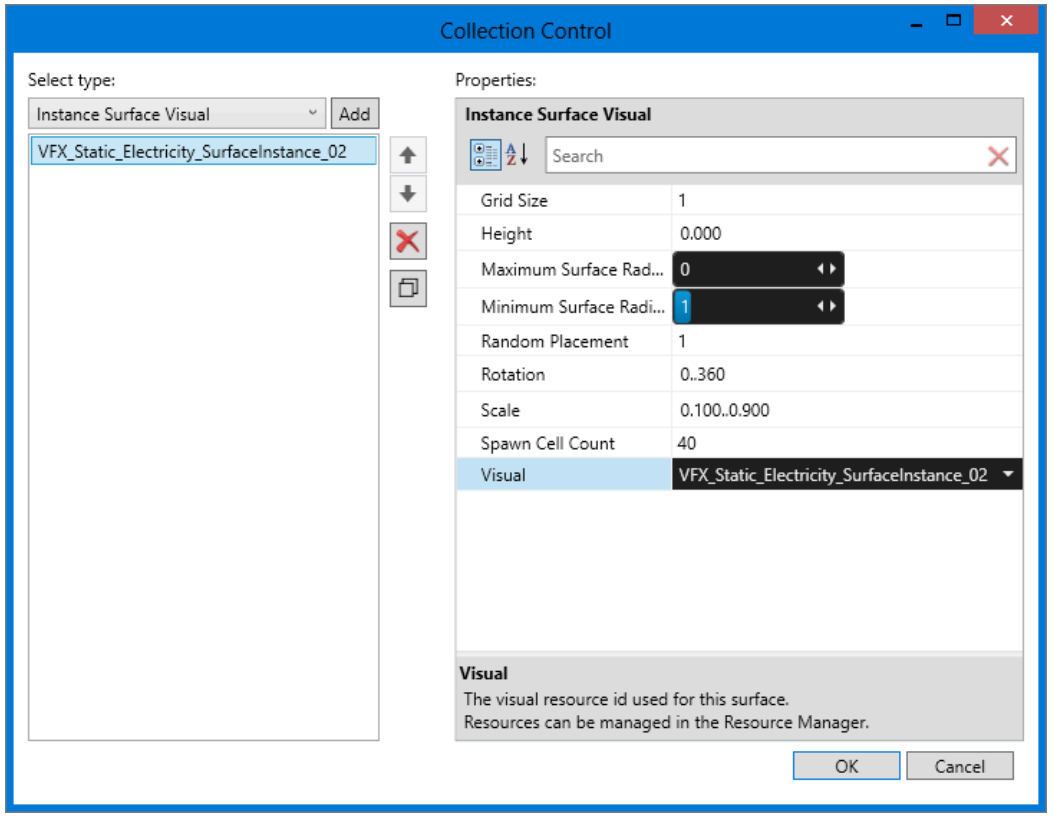
- **Decal Material** is the majority of a Surface's appearance. All the Surface Decals can be found in the Resource Manager by sorting by Materials (The bright blue sphere icon) and putting VFX_Surface into the Filters box.
    - There are some Decal Materials that exist for Surfaces like BloodFrozen, and many that are for Clouds (any which include "particles" in part of the name is probably for a cloud).
    - For some reason, most of the Cloud Decals don't actually do much, and use Instances for the majority of their visuals.
    - A couple have mismatched names, like VFX_Surface_Poop, for SurfaceSewer.
    - There are some variants accessible in the dropdown, like Surface_Blood_Electrified_Blessed, that have a different look to them, but do not appear in the Material Resource Manager.
    - IMPORTANT: Ground-Based Surfaces cannot display Cloud Decals and Cloud-Based Surfaces cannot display Ground Decals. This is HC.
- **Fade In Speed** and **Fade Out Speed** adjust how quickly the Surface appears and disappears, but the difference between 1, 5, and 10000 is so small as to be irrelevant. 5 works fine for both.
- **Icon** is the image representing the surface that displays in Tooltips. There are guides on creating icons on mod.io, so that is outside the scope of this guide.
- Instances are the visual extras that a Surface spawns. Whether it's bolts of electricity from SurfaceWaterElectrified or the thorns from SurfaceSpikeGrowth, it's defined here. A Surface can have multiple Instances. They will persist for the entire duration of the Surface. Click the ellipses (...) button to the right of the field to add or remove Instances in the window that opens. See SIDEBAR: Adding and Changing Instances and Effects for details.**Intro Effects** will play once, when the Surface is created. The visuals available here are different from the ones available in Instances.
- **Loop Effects** will play repeatedly, similar to Instances. The visuals available here are different from the ones available in Instances.
- **Material Type** tells the game what sound is played when an entity steps on this Surface, whether it should create splashes or footprints, and what those particles would look like. Clouds usually are set to Ghost.

- **Normal Blending Factor** will change a Ground-Based Surface's fade at the edge, and how much of the ground's texture can be seen through it. An entirely opaque liquid would be 0, while a partially translucent one may work well at 0.25 or so. Note that some decals are already transparent and thus have this set to 0.
- **Random Seed** will change how randomly puddles of the Surface grow. Common seeds are 0, 10, and 666.
- **Surface Grow Timer** is connected to puddles and Zone spells. If a Surface is told to grow 4 times, this setting determines how far apart (in rounds) those growth steps occur. Usually, Surfaces grow to their full size inside of a single turn, so this is usually something like 0.2 (which would be 1.2 seconds, making the Surface finish its 4 steps within 4.8 seconds).

Now that you've made a lot of changes, remember to Save your Surface. The Save icon in the Root Templates pane or at the top of the TK can both do this.

SIDEBAR: Adding and Changing Instances and Effects

While the Visuals for Instances and Effects come from different places, the window for adding and removing them has the same layout for Instances, Intro Effects and Looping Effects. Just be certain you've opened the correct one before continuing.



Adding, Removing, and Copying Instances and Effects works the same as it does for Status Data.

- **Grid Size** is the spacing between the visuals, as if you were setting the width of the invisible lines of the map grid. 0 to 4 works quite well for most visuals. Making this too large will create strange gaps in the visual's effect.
- **Height** is how high above the Surface the visual should appear. 0.00 is the ground, 1.00 is about chest height for a Medium character. Setting it to a single number makes all of the effects appear at that height. A random range can be specified with "#.##..#.##" replacing # with the desired heights.
- **Maximum Surface Radius** and **Minimum Surface Radius** are a scale from 0 to 10 defining how close to the center of the surface the visual can appear. Higher values don't always work very well.
  - If the Maximum is 0, it creates a pocket of the visual in the center of the Surface based on the Minimum.
  - If the Maximum is greater than the Minimum, the visual will appear on the outside edge of the Surface, expanding inward more based on the Maximum.
  - If the Maximum and Minimum are equal, or the Maximum is less than the Minimum but greater than 0, the visual will not appear.
  - If the Surface is a Cloud, setting both to 0 will still work for some reason.
- **Random Placement** will determine how far apart the visuals appear from one-another. It doesn't change much, so likely 1 is sufficient.
- **Rotation** is the range in degrees (0-360) that the visual could randomly be rotated. The syntax is "#..#" replacing the # with the low and high degrees respectively. Setting it to a single number makes all of the effects appear at that rotation.
- **Scale** is a range of size (0.000-1.000) that the visual can randomly spawn as. The size you choose probably depends on the visual itself, but should start larger than 0.00. While it can go as large as you want, it probably is best not to make it larger than 1.00. The syntax is "#.###..#.###". Setting it to a single number makes all of the effects appear at that size.
- **Spawn Cell Count** will determine how many of the visual should appear at once. The higher this number, the more "cells" of the Surface will create the visual. 40 creates the visuals fairly well-spread throughout the Surface.
- **Visual** is the actual effect you want to display. The drop down will list all visuals of the right type regardless of whether they were designed to be used for a Surface. The Resource Manager can be used to browse or add new visuals.

Step 4: Testing Your Surface

You've made all sorts of changes and want to see how it all looks and acts. It's time to make a test spell to create the Surface and take a look!

1. Making a Spell

It's probably easiest to start with a spell that already creates a similar Surface. For summoning Cloud-Based Surfaces, Target_FogCloud works well. For Ground-Based Surfaces, try Target_CreateWater.

1.1. Open the Stats Editor and put "Target" in the search bar. Both of these recommended spells are under Shared, so expand the sections under Shared > Spell Data to find Target and double-click to open it.

1.2. Use the Local Search (CTRL F) to find the spell by typing in "Fog" or "Water" and click on the correct search result.

1.3. In the large pane, select the entire row containing the spell by clicking on the number to the left of it. Then, right click and hover over "Create derived in" before clicking on the name of your mod which appears next to it.

1.4. You will be brought to the Target section of your own mod. Rename the new spell, using underscores instead of spaces. The editor will add "Target_" to the beginning in the code, so you won't need to include that.

1.5. Scroll to the right until you find column K "SpellProperties". It is recommended to widen the column by clicking and dragging the edge of the column far enough to see the contents of the cell fully.

1.6. Edit the text in that cell. CreateWater currently reads `ApplyStatus(WET,100, 3); GROUND:CreateSurface(4,0, Water);` Remove everything before the word "GROUND", and replace the word "Water" with the Name of the Surface (NOT the Display Name), leaving out the word Surface. It will likely read something like `GROUND:CreateSurface(4,0,BloodElectrified);`

1.7. If you wish, change the radius in meters (the first number after the parentheses) and/or the lifetime in rounds (the second number). 0 uses the default lifetime of the Surface, -1 is permanent.

1.8. To avoid confusion, you may wish to change the DisplayName of the spell, all the way over in column AB. This will be what the spell is called in-game.

1.9. Save your work by clicking the Save icon or using CTRL S. This may take a second or two.

2. Giving your spell to a character

You've got a spell that should summon your Surface, now you need someone who can cast that spell.

2.1. Returning to the main TK window, enter Game Mode . If you haven't saved your RootTemplate or Stats, you will be prompted to do so now. Make sure you press YES, as pressing NO will revert all your changes. Pressing CANCEL will not save and also will not revert any changes.

2.2. Press the F11 key to open the console window in the game pane. Click on the window's text box to be sure you have it active.

2.3. Type in the command `addDebugSpellAsClass` and press the TAB key. Then add the Name of your spell, including the "Target_" part at the beginning. Then press the TAB key. TIP: You can select the spell in the right panel of the console to speed up the process. The more of the name you type, the more it will refine the list.

2.4. Select or type the name of the class used by the character spawned by the TK.

2.5. Press ENTER.After following these steps, the spell should now appear in the left section of the hotbar, as you added it as a Class Action. This is to avoid needing Spell Slots to cast it. If you wish to have it work as a normal spell, use `addDebugSpell` in step 2.3 instead and skip step 2.4.

2.6. You can now close the console with the x button or F11 key.

3. ### Cast the spell
   Have the character cast the spell on an open area. Your Surface should appear. If it does not, try some of the Troubleshooting methods near the end of this guide.

Step 5: Implementing Your Surface

Alright, you now have a working Surface… but even if you want it to work only when your test spell is cast, the spell isn't accessible yet to any characters in-game, and your Surface will never be seen by players. Here are some suggestions as to how to have players interact with your Surface.

Spells

These sections will have some details specific to Surfaces, but otherwise may be familiar to you. To skip right to adding spells to the game, look at Giving Access to your Spell later in this guide.

Target

While it is likely the most common method, Target spells are a reliable way to have your Surface accessible in-game. Creating a Target spell is outlined above in Step 4 - Testing your Surface, under "Making A Spell". This is also how attacks like a Water Elemental's Slam Attack create a Surface.

Projectile

Besides the creation of spells akin to Acid Arrow or Fireball, you'll need to create a Projectile spell in order to implement any arrow or grenade. Any other explosion that results in an instance of your Surface will require a Projectile spell as well.

1. Open the Stats Editor and put "Projectile" in the search bar. The majority of the spells you'll likely want as a base are in Shared, so expand the sections under Shared > Spell Data to find Projectile and double-click to open it.
2. Use the Local Search (CTRL F) to find a good base spell by typing in "Arrow", "Grenade", or a word you'd find in a similar spell such as "ChromaticOrb". Click on the correct search result.
3. In the large pane, select the entire row containing the spell by clicking on the number to the left of it. Then, right click and hover over "Create derived in" before clicking on the name of your mod which appears next to it.
4. You will be brought to the Projectile section of your own mod. Rename the new spell, using underscores instead of spaces. The editor will add "Projectile_" to the beginning in the code, so you won't need to include that.
5. Scroll to the right until you find SpellProperties (column K). Depending on the base spell, you may only need to edit the Surface name portion of this cell. The syntax is `GROUND:CreateSurface([radius],[lifetime],[SurfaceName]);` The SurfaceName should NOT include the word Surface. If there are other features, like making other Surfaces Melt (even if the Surface might not, see the BoilingWater throwable in Shared) you can include or change that here too.
6. If you wish, change the radius in meters and/or the lifetime in rounds (0 uses the default lifetime of the Surface, -1 is permanent). For reference, most arrows that create Surfaces have a radius of 2 and a lifetime of 0 (or leave it blank). Most Grenades create a radius of 2 or 3 and many have a lifetime of 0, but if not, they are typically set to 3.
7. Change the DisplayName (column AD). This will be what the spell is called in-game.
8. Change the Description (column AE). This is what shows in the Tooltip for the spell.

9. (Optional) Change the damage done in SpellProperties, or if there's a full explosion, in SpellSuccess and/Or SpellFail (columns U and V).
10. (Optional) change the DamageType (column CZ) to one that closer matches your spell or the damage done by the spell.
11. Save your work by clicking the Save icon or using CTRL S. This may take a second or two.

Wall

Several Wall spells combine Surfaces, summoned objects, and cool visuals to create excellent battlefield control spells… and you can create your own!

SIDEBAR: Emulating SurfaceChange with Zones

The game can be tricked to have specific Zone spells create what appear to be SurfaceChange effects. In actuality, this identifies the location through SurfaceSentinel of another specific Surface or set of Surfaces in the Zone's range, then creates another Surface in its place while clearing the previous one.

Sentinel is an invisible, non-replacing Surface that sends information about the spell's area back to the spell, which is why this works.

It is important to note it behaves differently from SurfaceChange in that it *only works on the portions of the Surface within the Zone's area.* Other parts of the Surface not within the area will not change. Also, the area always starts from the creator's position.

To do this, set up the Zone spell as outlined in the instructions, but for the SurfaceType, select Sentinel (the last entry in the list) instead of your custom Surface.

Then, in SpellProperties, use the following code:

```
GROUND:IF(Surface('[SurfaceBeingChanged]'))
:CreateSurface(1,0,[SurfaceToChangeInto]);G
ROUND:IF(Surface('[SurfaceBeingChanged]')):
SurfaceClearLayer(Ground);
```

To fine-tune the changed Surface size, try changing the CreateSurface from 1 to 0.75 or lower, though smaller patches may not give as satisfying of a visual result. Setting higher than 1 will usually make the new Surface larger than the original.

Remember that SurfaceBeingChanged is in single quotes and does need the word "Surface" at the beginning, but that SurfaceToChangeInto will not have any quotes and does NOT have the word "Surface" in it.

Follow the Projectile instructions above, using Wall instead of Projectile. Note that the majority of Wall spells are in SharedDev instead of Shared. The columns will be in a different order. The following instructions will differ for Wall spells.

In step 5, instead of editing SpellProperties, scroll to SurfaceType (column T) and input the name of your Surface (still without the word Surface at the beginning) or select it from the dropdown.

In step 6, the radius for your Surface is the entry in SurfaceRadius (column U), still using meters. Walls usually keep their radius to 1 to match the size of the wall, or if it extends the Surface outward it is usually 2. The lifetime is defined in Lifetime (column S), still using Rounds.

The object spawned by the wall (such as the ice block part of Wall of Ice) is defined in ItemWall (column BH). The objects can have statuses defined in ItemWallStatus (column BI). Wall of Ice uses a status on its ice blocks to cause a small explosion when a block is destroyed, which in turn creates an instance of SurfaceIceCloud above the SurfaceWaterFrozen instance. Adding new wall objects is done in the UUID Object Editor under ItemWallTemplates. Creating those is outside the scope of this guide.

Zone

Not just for breath weapons! Zone spells can be used as a workaround to some of the restrictions your Surface suffers from. You can use Zone spells to emulate Vaporizing, Melting, Condensing (and more?) some Surfaces that otherwise would not do so, see SIDEBAR: Emulating SurfaceChange with Zones for details.

Follow the Projectile instructions above, using Zone instead of Projectile. The columns will be in a different order The following instructions will differ for Zone spells. Note that Zones will always expand from the creator's position.

In step 5, instead of editing SpellProperties, scroll to SurfaceType (column L) and select your Surface in the dropdown. If the Zone should remove other Surfaces, that can be done in SurfaceProperties with a CreateSurface of None.

Step 6 is completely different.

- Instead of using a Radius, the area of the Surface is defined across several other cells.
    - The shape of the area (cone or square) is set in Shape (column AX).
    - The distance (in meters) the Surface will cover is set in Range (column BB).
    - If the shape is Square, OR if you want the cone to start with a larger width, set the width in Base (column BC).
    - For Cones, the angle of the cone is defined in Angle (column BD) and can be any degree you wish between 1 and 360.
    - If the Zone should expand from closer or further from the creator, set that distance in Offset (column AY), with a negative number for closer and a positive one for further. An Offset of -2 starts from the center of the caster.
- The Lifetime is defined in SurfaceLifetime (column M), and still uses Rounds, with 0 for Default Lifetime and -1 for Permanent..
    - The speed at which the Surface expands to cover the area is defined with SurfaceGrowStep (column N) as a percentage (0 - 100, no percentage symbol) representing how much of the Surface will appear in each step.
    - The number of the steps is set in SurfaceGrowInterval (column O). Setting this to 1 means the Surface will fill the area by the percentage in SurfaceGrowStep once by the time the casting is done. If the percentage is less than 100, it will not fully fill the area as it will not have enough steps to do so. A percentage of 25 would need 4 steps to fill the entire area. If the percentage is larger than the steps needed (for example, percentage 25 and 5 steps), the extra steps will not continue expanding the Surface.

Giving Access to your Spell

To allow players to actually cast any of the Spells you've just made, you'll need to either add it to a Class Spell List or add the spells to Items or Boosts.

- To add to a Class Spell List:
    - Open the UUID Object Editor
    - Find the Spell List of the Class you wish to add the spell to, and note its name. They are found in the Lists category, with most being under Shared or SharedDev.
    - Create your own SpellLists in the Lists category of your own mod.
    - Create a new Spell List by filling out the Name cell. It will automatically create a UUID.
    - Put the name of your spell, including the preface (Target_, Projectile_, etc) in the Spells cell. Multiple spells can be added with a semicolon (;) separating them.
    - In the MergedInto cell, use the dropdown to select the name of the Spell List these spells will be added to.
    - Save your work.
- To add only to a specific entity:
    - Find its Root Template and open it. Override it if necessary.
    - Locate the Spells field of the Root Template, and click the ellipses (...) to open the Spells window.
    - Filter the list for your spell, select it, and use the right double carrot button (>>) to add it to the entity.
    - You can edit minor details about how the entity should use the spell by selecting it and editing the entries in the right-hand field
    - Save the Spells, then Save the Root Template.
- To add to a Boost or Item, include the line `UnlockSpell([YourSpellName]);` in the Boosts or BoostsOnEquip field. This will add the Spell to the character affected by the Boost or equipping the Item. See Boosts below for more details.
- To have a Boost, Item, Passive, or even another Spell create an explosion, create a Projectile spell and reference it with `CreateExplosion(Projectile_[YourSpellName])`.

Blood

Any entity can bleed your Surface when injured or killed, and it is easy to do! Here's how:

1. Open the RootTemplate of the entity (override it if you need to).
2. Locate the Blood Surface Type in the Death section and expand the drop down menu. Find the Name of your Surface and select it.
3. Optional: find the BloodType in the Hit VFX section. This controls what color the blood that sprays out of a character when injured is. Select a color similar to your Surface.
4. Save the RootTemplate.

That's all it takes! A puddle of your Surface will now appear when that character is injured or killed, or when the object is destroyed (depending on what kind of entity it is). Any other entity using this entity as a Parent will create such a puddle as well, as long as it doesn't also override this setting.

Altered Blood

While an entity cannot technically have its blood changed from the [setting in its Root Template] mid-game, the Osiris code that exists to handle the Unstable Blood ability that can be gained in Act 3 does an acceptable job of emulating that change. That code is explained below, but this guide will not be going into detail on how Osiris works. Some extra information on the use of Osiris specifically for Surfaces can be found [later in this guide.]

```
INIT
DB_LOW_BloodMerchant_DamageWithBlood("Slashing");
DB_LOW_BloodMerchant_DamageWithBlood("Piercing");
DB_LOW_BloodMerchant_DamageWithBlood("Bludgeoning");

KB
IF
AttackedBy(_Player, _, _, _Damage, _, _, _)
AND
DB_LOW_BloodMerchant_ExplosiveBloodCharacter(_Player)
AND
DB_LOW_BloodMerchant_DamageWithBlood(_Damage)
THEN
CreateSurface(_Player, "SurfaceBloodExploding", 2.0, -1.0);
```

The code above triggers anytime an entity who is listed in the `DB_LOW_BloodMerchant_ExplosiveBloodCharacter` Database takes Slashing, Bludgeoning, or Piercing damage. It then creates a 2-meter Radius Surface of Unstable Blood, with a permanent duration. This does NOT stop the entity from bleeding their normal blood, but rather creates SurfaceBloodExploding in a large enough area to cover the entity's normal blood spatter. Note that while `CreatePuddle` would work more like their normal blood, it would be too random to reliably cover the SurfaceBlood they would normally spill. Also note that any effect that causes them to bleed without dealing Slashing, Bludgeoning, or Piercing damage would still create normal blood as well, since no other damage types are listed in the Database creating in the INIT section of the code.

Items

Many Surfaces come exclusively from Items. There are a lot of easily-implemented options, and with some extra effort, some very interesting potential creations.

Arrows

Fairly simple to implement and deploy, gives any character that can use a bow access to your Surface. There are three main steps, creating a Projectile Spell, creating a RootTemplate, and creating an Object Stat entry.

1. Creating a Projectile Spell
    1.1. Follow the instructions in the Projectile portion of the Spells section above.
2. Creating a Root Template
    2.1. Find a base for your Arrow in the Root Templates pane, right click its entry, and select “Create inherited from selected”.
    2.2. A window will open like the one we saw while overriding a Surface. This time, we don’t want to override, so give it a new name. It is best to follow the naming conventions of other arrows, like ARR_Arrow_Of_[Surface] and Save.
    2.3. Edit the Display Name, Description (Lore), and Icon entries to appropriate choices.
    2.4. Edit the Use Action by clicking the ellipses button (...) to the right of the field. See SIDEBAR: Editing Use Action to add a Spell or Throw Action to an Object for details.
    2.5. Find the Stats field and press the Open button to the left of the field, then continue on to step 3.
3. Creating an Object Stats Entry
    3.1. If you have just come from Creating a Root Template, move to step 3.2. Otherwise, find a base Stats entry from another arrow in the Object Stats of Shared or another directory.
    3.2. Select the entire row of the base arrow’s stats and right click it. Hover over the Create derived in, then select your mod’s name where it appears to the right.
    3.3. Change the Name (Column A) to an appropriate one. Again, naming conventions are helpful, so OBJ_ArrowOf[Surface] works well.
    3.4. Change the RootTemplate (Column C) to match your new Root Template. If you can’t get it to change correctly, go to the main TK window and Save, then reopen the Stats window and try again.
    3.5. Save and Export or press CTRL S.
    3.6. Back in the RootTemplate, find and select the Stat entry you just created. If it won’t appear, briefly select another RootTemplate, then switch back to the one for your arrow and try again.
    3.7. Save the RootTemplate.

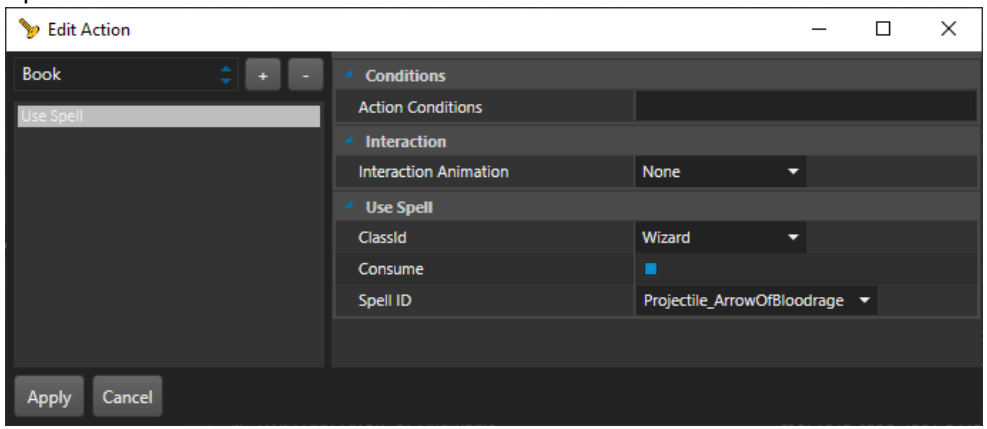
Congratulations, you’ve created an arrow for your Surface! You’ll see it sold by any shopkeep that sells arrows of the correct rarity. To change what rarity it is, open the Stats entry and edit Rarity (column K) and ObjectCategory (column Y). If you wish to see it as loot, you’ll need to add it to Treasure Tables.

A little more difficult than implementing an arrow, but gives most characters access to your Surface. There are three main steps, creating a Projectile Spell, creating a RootTemplate, and creating an Object Stat entry. If you wish to have the grenade be craftable with Alchemy, that is an additional step that is outside the scope of this guide.

1. Creating a Projectile Spell
    1.1. Follow the instructions in the Projectile portion of the Spells section above.
2. Creating a Root Template
    2.1. Find a base for your Grenade in the Root Templates pane, select its entry, then right click it and select “Create inherited from selected”.
    2.2. A window will open like the one we saw while overriding a Surface. This time, we don’t want to override, so give it a new name. It is best to follow the naming conventions of other grenades, like GRN_[Surface] and click Create.
    2.3. Edit the Display Name, Description (Lore), and Icon entries to appropriate names and choices.
    2.4. Edit the Scripts by clicking the ellipses button (...) to the right of the field. A window will open.
        2.4.1. If you are using another Grenade as the base, it should have an entry in the center “Assigned” pane that reads “DOS2_Grenade”. If not, add it by finding that entry in the left “Available” pane, selecting it, then using the right double carrot button (>>) to add it to the “Assigned” pane.
        2.4.2. Select the “DOS2_Grenade” entry in the Assigned pane.
        2.4.3. In the pane on the right, under Globals, find the ProjectileSpell field and edit it to the Projectile spell you created.
        2.4.4. Press “OK”
    2.5. Edit the Use Action by clicking the ellipses button (...) to the right of the field. See SIDEBAR: Editing Use Action to add a Spell or Throw Action to an Object for details.
    2.6. Find the Stats field and press the Open button to the left of the field, then continue on to step 3.
3. Creating an Object Stats Entry
    3.1. If you have just come from Creating a Root Template, move to step 3.2. Otherwise, find a base Stats entry from another grenade in the Object Stats of Shared or another directory.

SIDEBAR: Editing Use Action to add a Spell or Throw Action to an Object

This window has a list of the actions it gives access to when Right Clicked. The topmost one will be the default when double-clicked instead.



1. If it does not already have an appropriate entry, expand the top left dropdown and select Use Spell or Throw, as appropriate, then press the + button.
2. For Use Spell:
    a. Leave Action Conditions blank.
    b. Set Interaction Animation as None.
    c. ClassId doesn’t matter for this. Setting to Wizard for consistency is fine.
    d. Make sure the Consume box is checked.
    e. Open the dropdown and find the Projectile spell you created earlier for this.
3. For Throw:
    a. Leave Action Conditions blank.
    b. Set Interaction Animation as OBJ_Throw_01.
    c. Set the Fallback Preview Radius to the expected Radius the object will fill with your Surface, or 0 if you aren’t sure.
4. Press Apply.

3.2. Select the entire row of the base grenade's stats and right click it. Hover over the Create derived in, then select your mod's name where it appears to the right.
3.3. Change the Name (Column A) to an appropriate one. Again, naming conventions are helpful, so OBJ_[Surface]_Bottle works well.
3.4. Change the RootTemplate (Column C) to match your new Root Template. If you can't get it to change correctly, go to the main TK window and Save, then reopen the Stats window and try again.
3.5. Save and Export or press CTRL S.
3.6. Back in the RootTemplate, find and select the Stat entry you just created. If it won't appear, briefly select another RootTemplate, then switch back to the one for your grenade and try again.
3.7. Save the RootTemplate.

Congratulations, you've created a grenade for your Surface! Depending on the base grenade, you *might* see it sold by any shopkeep that sells grenades of the correct rarity. To change what rarity it is, open the Stats entry and edit Rarity (column K) and ObjectCategory (column Y). If you wish to see it as loot, you'll need to add it to Treasure Tables.

Armor, Clothing, and Accessories

Using a Boost or Passive, anything from a ring, to shoes, to a helm can give a person intriguing new ways to play with your Surface.

"Armor" is the general category that includes *anything in the equipment slots*, *except for weapons and torches.* Amulets, instruments, even shields- if it is worn, it is Armor as far as the game is concerned. Armor is found in the Stats Editor under the Stats category. If you are creating Armor from scratch, you'll need to also create a Root Template, similar to the Arrows or Grenades mentioned above.

To add a Spell to anyone wearing a particular Armor item, put `UnlockSpell([YourSpellName]);` in the BoostsOnEquip cell (column AC).

To add a Boost to the Armor, add the name of the Boost to the DefaultBoosts cell (column AA). This Boost affects the Armor itself, not the wearer. To have the entity wearing the Armor be affected by the Boost instead, use the StatusOnEquip cell (column AE).

Passives given to the entity wearing the Armor can be added in PassivesOnEquip (column AD). To add a Passive to the Armor itself, attach the Passive to a Boost and use the DefaultBoosts cell.

If you want the Armor to impart a Boost to anyone carrying it *even while it is not equipped*, use the StatusInInventory cell (column BC).

Weapons

Anything from a bow to a sword that creates your Surface can be a boon or a bane to the wielder, depending on your Surface and how the weapon creates it. Note that "Weapons" includes torches, despite using a different equipment slot, and shields are considered Armor despite filling a weapon slot. Weapons are found in the Stats Editor under the Stats category. If you are creating a Weapon from scratch, you'll need to also create a Root Template, similar to the Arrows or Grenades mentioned above.

To add a Spell to anyone wielding a particular Weapon item, put `UnlockSpell([YourSpellName]);` in the BoostsOnEquip cell (column AJ). Alternatively, the Spell can be given depending on whether the Weapon is equipped in the Main Hand using BoostsOnEquipMainHand cell (column AK) or Off Hand using BoostsOnEquipOffHand cell (column AL).

To have the Weapon only activate when attacking, use the WeaponFunctors cell (column AF) as if it was the SpellProperties cell of a Spell. `SurfaceChange` and `CreateSurface` are good choices for creating your Surface with a Weapon.

To add a Boost to the Weapon add the name of the Boost to the DefaultBoosts cell (column AH). This Boost affects the Weapon itself, not the wielder. To have the entity wielding the Weapon be affected by the Boost instead, use the StatusOnEquip cell (column AP).

Passives given to the entity wielding the Weapon can be added in PassivesOnEquip (column AD). Alternatively, the Spell can be given depending on whether the Weapon is equipped in the Main Hand using PassivesMainHand cell (column AN) or Off Hand using PassivesOffHand cell (column AO).To add a Passive to the Weapon itself, attach the Passive to a Boost and use the DefaultBoosts cell.

If you want the Armor to impart a Boost to anyone carrying it *even while it is not equipped*, use the StatusInInventory cell (column BC).

Other Objects

Items besides those above can be useful of course. This includes Scrolls, one-time or multi-use summoning items, and books. Most can be made similarly to Arrows and Grenades. Consider options such as adding `InSurface([YourSurfaceName])` to the Requirements cell (column P) to restrict its use to only while in your Surface, or adding a Boost to StatusInInventory (column BF) to bind items to an entity, like the Mysterious Artifact.

Adding Items to Treasure Tables

- Open the Stats Editor .
- Find the Treasure Table of the entity you wish to add the item(s) to, and note its Name. They are found in the TreasureTable category of Shared, SharedDev, Gustav, and GustavDev.
- Create a new TreasureTable in the TreasureTable category of your own mod. Give it the same Name (column A) as the one you wish to add your items to.
- Set the AppendNotOverwrite (column B) to **Yes**. Otherwise, you will remove all other items from that container!
- Put the Name of your item in the Treasure cell (column I) AND add I_ to the beginning of the Name, syntax of `I_[Your_ItemName].`
- Set the DropCount (column H) to 1,1. This means 1 Item in 1 stack of items. To make random amounts spawn, include multiple variant numbers in that cell with semicolons (;) separating them (for example 1,1;2,1;4,4).
- Set the Frequency to 1.
- To put another item in the same container, right click the first row of that container and select “Add Sub Stat” and fill that out the same way as the first item, but leave the Name and AppendNotOverwrite cells blank for each item in that same container.
- Save your work.

Boosts

While adding a Boost to an Item is how Boosts are often used, they are also what will give your Surface interesting effects. Specifically, Boosts are what the Status Data field in the Gameplay section of the Surface's Root Template is referencing.

Boosts can be created in the Stats Editor . Unlike spells, Boosts won't be found in the SpellData category, but rather under the StatusData category. For most Boosts you will want to use a Status_BOOST.

A Boost can apply Passives (column AZ) to entities it affects, as well as other Boosts (column AY). It can create an Aura (column AL) which is great for having Surfaces appear within a particular area or to follow an entity (disappearing from anywhere they move away from, unlike a Trail). You can even use them as a way to Ignite or otherwise change applicable Surfaces as per SurfaceChange (column AO).

The Boosts field is not restricted to using other Boosts. It will also operate similarly to the SpellProperties field in a Spell. Likewise, the TickFunctors field (column BE) works the same, except that its effects will only trigger once per turn based off of the TickType field (column AX). Consider some options like `IF(InSurface([SurfaceName]))`, or `CreateExplosion([Projectile_YourSpell])` referencing a Projectile spell that creates your Surface, or even something like `DealDamage(1d4,Fire)` to be added to your Surface to apply damage to those who enter.

If the Boost is intended as an effect your Surface applies, remember that in addition to TickType, you also have control of when the Status using that Boost takes effect inside the Status Data field. Also, it is recommended to add `SG_Surface` in the StatusGroups field (column BS) so that the game will properly recognise it as being connected to the Surface.

If you want something to happen whenever an entity attacks, that is typically handled by the Weapon itself. If you want something to occur when an entity is damaged by something, that is typically handled by Passives.

Passives

As long as it is active, a passive can do some fairly crazy things. Usually this is through constant Boosts, but there are some alternatives to consider.

A Passive can be created in the Stats Editor . Unlike Boosts, Passives won't be found in the StatusData category, but rather under the Stats category. Passives often use Boosts, so being familiar with those is very helpful here. For example, Auras are created with Boosts, not Passives, but a Passive can give an entity a Boost with an Aura, making it seem as if the Passive was providing it.

Likely the most useful part of Passives will be the Toggle settings (columns Y through AD). Specific to Surfaces, the EnabledContext cell (column Q) has an option to Enable the Passive when entering a Surface by checking the box for `OnSurfaceEnter`. Which Surfaces to check for are defined in EnabledConditions (column P). Similar functionality (but for Boosts given by the Passive) can be found in StatsFunctorContext (column V) and StatsFunctorConditions (column w)

Also, Passives can apply Boosts, and Boosts can apply Passives, so you can create a Boost that gives a Passive that gives a different Boost, that in turn gives a different Passive, and so on.

Trails

While this requires some Osiris coding, it can turn your Surface from a singular patch into a barrier, fuse, ground writing method, or simply gives some personalization to a creature.

Achieving this requires Osiris coding, and this guide will not be going into detail on how Osiris works. However, some extra information on the use of Osiris specifically for Surfaces can be found later in this guide.

You will also need the name and UUID of the entity that will spawn the trail.

The script can be located at the end of the normal scripts or sectioned off if it is only relevant in a particular part of the game.

- INIT only needs input if you have a list of different triggers or other info.
- KB should include two main parts, one to create the trail, and one to stop creating it.
- Note that `DrawSurfaceOnPath` is NOT a Call, but a Query that acts like a Call. `StopDrawSurfaceOnPath` is a Call as you might expect. More details on these are contained in the Osiris section later in this guide.
- With the correct Event or Query, this behavior could be tied to a Passive. Here is a fairly simple custom example using a Passive that can be toggled On and Off, which has a Boost named SURFACE_BLOOD_TRAIL added and removed in its ToggleOnFunctors and ToggleOffFunctors.

```
IF
StatusApplied((GUIDSTRING)_Object, "SURFACE_BLOOD_TRAIL", _, _)
AND
DrawSurfaceOnPath(_Object, _Object, "SurfaceBlood", 0.5, 30.00, (INTEGER64)_SurfaceActionHandle)
THEN
DB_SurfaceHandle(_Object, _SurfaceActionHandle);

IF
StatusRemoved((GUIDSTRING)_Object, "SURFACE_BLOOD_TRAIL", _, _)
AND
DB_SurfaceHandle(_Object, (INTEGER64)_SurfaceActionHandle)
THEN
NOT DB_SurfaceHandle(_Object, _SurfaceActionHandle);
StopDrawSurfaceOnPath(_SurfaceActionHandle);
```

    - The above example triggers on any entity that has the SURFACE_BLOOD_TRAIL Boost and makes the entity start creating a trail, outputting the handle for that trail to a database. Then, when the Passive is toggled off (which removes the Boost) for that specific object, the Database pulls up the handle connected to it, removes that entry from the database, and tells it to stop creating the Surface. This would also occur if the Passive was removed from the entity entirely (as it would also remove the Boost).

- Here is a more complicated example from the floating orb that burns the ground when you upset the mirror in the Act 1 alchemist basement.

```
IF
EntityEvent(S_FOR_ThayanOrb_c9b5f985-466d-41d4-88fd-adfa999aaeb1, "FOR_ThayanCellar_OrbEmerged")
AND
DrawSurfaceOnPath(S_FOR_ThayanOrb_c9b5f985-466d-41d4-88fd-adfa999aaeb1,
SS_FOR_ThayanOrb_c9b5f985-466d-41d4-88fd-adfa999aaeb1, "SurfaceFire", 0.5, 30.0, _Handle)
THEN
DB_FOR_ThayanCellar_SurfaceHandle(_Handle);
ApplyStatus(S_FOR_ThayanOrb_c9b5f985-466d-41d4-88fd-adfa999aaeb1, "FOR_THAYANCELLAR_TRAPFUEL", 60.0, 1,
NULL_00000000-0000-0000-0000-000000000000);
PROC_FOR_ThayanCellar_OrbTrace();

IF
TimerFinished("FOR_ThayanCellar_ReturnDelay")
AND
DB_FOR_ThayanCellar_SurfaceHandle(_Handle)
THEN
NOT DB_FOR_ThayanCellar_SurfaceHandle(_Handle);
StopDrawSurfaceOnPath(_Handle);
RemoveStatus(S_FOR_ThayanOrb_c9b5f985-466d-41d4-88fd-adfa999aaeb1, "FOR_THAYANCELLAR_TRAPFUEL");
ItemMoveTo(S_FOR_ThayanOrb_c9b5f985-466d-41d4-88fd-adfa999aaeb1,
S_FOR_ThayanOrbEmerge_338a236a-feb1-473b-8b69-0bc6b86dcad1, 1.2, 0.0, 0, "FOR_FireOrb_LeavePath", 1);
```

    - In the above example, when the Event occurs, the orb gains a Boost, immediately begins creating the Surface wherever it moves, and the handle connected to that behavior is saved to a Database. When a timer that was set in another section of code (not displayed here) ends, the orb loses the Boost, stops creating a Surface and removes the handle from the Database. Of note, the Boost itself may have a connection to the timer, but otherwise would not have any impact on the creation of the Trail and is included to give the players information about what the orb is doing and for how long.

Naturally or Superficially Spawning

Your Surface might be found out in the game world as part of nature, or could have been superficially created by some magical effect from before the players arrived. Examples include the patches of SurfaceFire (supposedly from Alchemist Fire bottles) found near the gnolls in act 1, areas of SurfaceMud in many places throughout the game, or the sections of SurfaceSewer in the act 3 sewers.

Achieving this requires Osiris coding, and this guide will not be going into detail on how Osiris works. However, some extra information about the use of Osiris specifically for Surfaces can be found later in this guide.

You will also need either the X, Y, *and* Z coordinates of the location you wish the Surface to appear, or the name and UUID of an object, character, or Trigger that can act as the center or source of the desired location. Check this other guide for details on finding coordinates.

The script should be located with the scripts of the correct Act that holds the location(s), and if defined separately, the Region within that Act. You would need multiple files for multiple different locations or Acts.

- INIT for each file should contain database entries for each target or set of coordinates. Optionally, you can include the Surface's intended size (in meters, as a Real number) at that location, or in the case of multiple different kinds of Surfaces, the Surface's Name as a string. Possible syntax: `DB_[SURFACE]_SPAWN([NAME_00000000-0000-0000-0000-000000000000], [SIZE]);` or `DB_MYSURFACE_SPAWN([X],[Y],[Z],[SIZE],[SURFACENAME]);`
- KB needs an identifying Event, such as when the player enters the general area. If you know the Trigger for an area, using `EnteredTrigger` works well. You can also try `LevelLoaded` if you know the level's map name. See this example for spawning Surfaces near Flind in Act 1.

```
IF
EnteredTrigger(_Player, S_PLA_Plains_SUB_bcc3efa5-bfe1-4224-9df1-6758cac764cd)
AND
DB_PartyMembers(_Player)
AND
DB_GlobalFlag(PLA_ConflictedFlind_State_NotEncountered_c4bed850-2d45-4806-bc6d-5d778e60cde4)
AND
NOT DB_GlobalFlag(PLA_ConflictedFlind_State_Gone_4152beef-e7d1-e10f-7b06-62584d3571f2)
AND
QRY_OnlyOnce("PLA_ConflictedFlind_Surfaces_Oneshot")
THEN
CreateSurface(S_PLA_ConflictedFlind_EntranceSurfaces_1_3f32deb8-6635-4ed4-9b6f-26184d351f65, "SurfaceFire", 1.5, -1.0);
CreateSurface(S_PLA_ConflictedFlind_EntranceSurfaces_2_256722b2-71d2-4bee-a619-6437d04957a8, "SurfaceFire", 1.5, -1.0);
CreateSurface(S_PLA_ConflictedFlind_EntranceSurfaces_3_032ad108-973d-4bba-9a52-cfc5ec2ea7e7, "SurfaceFire", 1.5, -1.0);
CreateSurface(S_PLA_ConflictedFlind_Zhentarim_Corpse_01973f38-1fd2-4035-abb3-27429f49bd47, "SurfaceBlood", 1.7, -1.0);
CreateSurface(S_PLA_ConflictedFlind_BurnedHyenaTrigger_15588ab4-e483-43ab-a52f-31880ab61ce9, "SurfaceFire", 1.5, -1.0);
```

  - In the above example, `DB_PartyMembers` is used to identify the Players but is defined in another script.
  - The GlobalFlags are not needed, and are only used in this example to check for the conditions to spawn the Surfaces, likely to account for entering Act 2 and then attempting to return to Act 1.
  - Note the use of `QRY_OnlyOnce` to keep from repeatedly spawning Surfaces every time the Trigger is entered.
  - The lifetime here is set to -1.0 (permanent) to ensure the Surface does not disappear before the player arrives.

- Another possible script example:

```
IF
Opened(TUT_Chest_Potions_e57e3af6-ae79-4d5c-9d11-f695b359c740)
AND
DB_BLOODRAGEMIST_SPAWN(_X, _Y, _Z, _Size, _SurfaceType)
AND
QRY_OnlyOnce("TUT_BloodrageMist_Surface_Oneshot")
THEN
```

`CreateSurfaceAtPosition(_X, _Y, _Z, _SurfaceType, _Size, -1.0);`

- This example would trigger when the well-renowned Tutorial Chest is opened, spawning Surfaces once (and only once, thanks to `QRY_OnlyOnce`) at locations specified in the INIT section using the `DB_BLOODRAGEMIST_SPAWN` Database. This would allow the Surfaces to appear in areas the player had already passed through, perhaps as an optional reward (or punishment) for opening the chest.

See the table of Calls in the Osiris section of the Extra Tips later in this guide for more options besides CreateSurface.

Extra Tips, Tricks, and Information

AI Interactions

- Character AI behaviors on or near the Surface are defined in the Stats entry for that character, in column Y “PathInfluence”. This is primarily what causes characters to avoid walking into dangerous ones and how the AI determines it is probably okay to enter others. The higher the number, the more the character will avoid it, with 10 being something like “barely hesitate to enter” and 100 being “avoid as much as possible”. This number is often multiplied by different amounts (to halve or double, for example) in different situations as defined in a character’s Anubis setting. This may actually cause some characters to purposefully enter what it believes to be a beneficial Surface. Note that Lava is usually set to the seemingly absurd 700, which even when reduced by 85% would in theory result in “DO NOT ENTER AT ANY COST, EXTRACT YOURSELF IMMEDIATELY.” Changing the _Base entry will change any character that uses that entry, or an entry based off of it, unless that character was already overriding it. The entry _Hero is the base for the Player’s path. A Surface not listed in this column may cause a character to act as if the Surface isn’t there.

- If a spell or other effect would create a Surface, the AI may not know when to use the spell or otherwise create the Surface. Putting `AI_IGNORE:` before the `GROUND:` portion will prevent the AI from taking into account the Surface’s creation. You can also provide the AI with fake information which it uses to make its casting decision. Adding `AI_ONLY`: right before this fake info will make the AI think that is the function of the spell without actually adding that to the spell, so `AI_ONLY:ApplyStatus(SILENCED,100,0);` would tell the AI that the Surface will Silence those within, even if that is not true. This would cause it to target spellcasters more often, or to avoid casting it on fellow spellcasters.

Tags

SURFACE_INTERRACTIONS_FROM_CENTER

described as “Do not use the pivot but the bound center as source of any surface transform actions caused by this object”

IGNORE_DANGEROUS_SURFACES

described as “Characters with this tag will not try and move out of dangerous surfaces. Be careful with this tag!”

SLIPPING_ADV

described as “Advantage on rolls against slipping surfaces.”

Spell and Effect Commands

In SpellProperties, Boosts, and Functor fields, you can use many different commands. Most will require `GROUND:` to be placed before it, as that is what is actually creating the Surface or making the change. Here are some of the more useful ones:

Functor	**Notes**
`CreateSurface([radius], [duration], [name], [(opt)concentration_boolean])`	Name should exclude the word Surface and does not use single or double quotes. Using the name `None` will remove all Surfaces in the radius instead. The optional Concentration setting will make the Surface disappear if the spell uses Concentration and the caster stops concentrating. Otherwise, the Surface should remain even if Concentration ends.
`CreateConeSurface([radius], [duration], [name], [(opt)concentration_boolean])`	ALWAYS originates from the caster in a cone of an estimated 15 degrees. Otherwise works like `CreateSurface`.
`SurfaceClearLayer([(opt)layer1], [(opt)layer2])`	Options for layers are ground and cloud, and do not use single or double quotes. Putting nothing in the parentheses clears all layers, equivalent to `SurfaceClearLayer(ground, cloud)`. The radius is defined by the spell's area. To define a different radius, use CreateSurface with None as the name (clears both layers). To define a different radius and only clear a specific layer, Osiris is required.
`SurfaceChange([change], [(opt)chance])`	Change options are `None, Ignite, Douse, Electrify, Deelectrify, Freeze, Melt, Vaporize, Condense, DestroyWater, Clear, Daylight, TurnHellfire`, and `UnturnHellfire`, and does not use single or double quotes. Use a separate SurfaceChange for each type. This only works if a Surface has a compatible HC transformation type. Chance is a percentage as a whole number, 1-100. If no chance is specified, 100 (guaranteed change) will be used. `Daylight` is not actually connected to a Surface, but exists for helping entities understand the source of magically-created sunlight.
`IF(CanTransformSurface([change], [(opt)position], [(opt)source])):`	Untested. Might work by using the same Change options as `SurfaceChange`, checking if target position or optionally provided position would respond to that change. Alternatively, might check whether a Boost or other effect causes a Surface Change.
`IF(Surface('[SurfaceName]', [(opt)target], [(opt)source])):`	Checks the Surface at the target point, at the optionally specified target, or within the spell's area. This DOES need the word Surface at the beginning of the Surface's name. The action to take depending on the IF comes after the colon. Also note the single quotes, i.e. `Surface('SurfaceBloodFrozen')`. The optional Source is untested at this time.
`IF(InSurface('[SurfaceName]'), [(opt)position], [(opt)check_proxy_bool]):`	Checks if either the caster or target (depending on context), or the optional position are within the specified Surface. Otherwise works as above.
`IF(IsWaterBasedSurface()):`	Works like `IF(Surface())` without the optionals, but checks for any of the these Surfaces all at once: `SurfaceWater, SurfaceWaterElectrified, SurfaceWaterFrozen, SurfaceBlood, SurfaceBloodElectrified, SurfaceBloodFrozen, SurfacePoison, SurfaceAlcohol, SurfaceBloodSilver, SurfaceWaterCloud, SurfaceWaterCloudElectrified, SurfacePoisonCloud, SurfaceFogCloud`
`IF(IsDippableSurface()):`	Works exactly like the `IsWaterBasedSurface` above, except it checks for these Surfaces: `SurfaceFire, SurfaceHellfire, SurfacePoison, SurfaceWater, SurfaceSerpentVenom, SurfaceWyvernPoison, SurfacePurpleWormPoison`
`IF(IsInElectrifiedSurface([entity])):`	Works like `IF(InSurface())` without the optionals, but checks for any of these Surfaces all at once: `SurfaceBloodElectrified, SurfaceDaggersCloudElectrified,SurfaceWaterCloudElectrified, SurfaceWaterElectrified`
`IF(HasDuration(StatsFunctorType.CreateSurface, [lifetime])):`	Looks through the SpellProperties of spells the player has and identifies any that create a Surface with a duration equal to or greater than the lifetime (or any lifetime if set to 0). Could be used in the Boost field for Metamagic, using `UnlockSpellVariant` instead of an IF statement. If you know how to specify a particular Surface, please inform the author of this guide.

`IF(HasFunctor(StatsFunctorType.CreateSurface)):`	As above, but does not check for a duration. Can be used on any of the other conditions above by replacing `CreateSurface` with the appropriate functor.

Osiris Events, Queries, and Calls

Using Osiris can further enhance your spells, effects, and more. While the specifics of writing code in Osiris are outside the scope of this guide, these lines are provided for ease of access.

Events

No Osiris Events directly reference Surfaces. However, it should be noted that many Queries and Calls will require either a Target or a set of coordinates to work, so Events such as `UseSpellAtPosition`, `UsingSpellOnTarget`, or `EnteredLevel` (on a Helper) are often helpful.

Queries

There are a lot of very useful queries regarding Surfaces. If a Query provides a Surface as a result, it will be the Name as a String. If you need a Target and there would not be one normally on the Surface, consider creating a Helper in a previous step.

Query	**Notes**
`GetSurfaceCloudAt([in](GUIDSTRING)_Target, [out](STRING)_Surface)`	Identifies the Surface located at the Target. Only checks the Cloud layer.
`GetSurfaceCloudOwnerAt([in](GUIDSTRING)_Target, [out](GUIDSTRING)_OwnerCharacter, [out](GUIDSTRING)_OwnerItem)`	Identifies the creator of the Surface located at the Target. Only checks the Cloud layer. Reports the UUID of the entity that “owns” the Surface.
`GetSurfaceGroundAt([in](GUIDSTRING)_Target, [out](STRING)_Surface)`	Identifies the Surface located at the Target. Only checks the Ground layer.
`GetSurfaceGroundOwnerAt([in](GUIDSTRING)_Target, [out](CHARACTER)_OwnerCharacter, [out](ITEM)_OwnerItem)`	Identifies the creator of the Surface located at the Target. Only checks the Ground layer. Reports the UUID of the entity that “owns” the Surface.
`GetSurfaceNameByTypeIndex([in](INTEGER)_SurfaceIndex, [out](STRING)_SurfaceName)`	Takes a whole number between 0 and 75 which represents the enumeration of a Surface in ValueLists, and returns the name of the Surface. Numbers outside of the expected range return an error.
`GetSurfaceSize([in](GUIDSTRING)_Target, [in](STRING)_SurfaceLayer, [out](INTEGER)_SurfaceSize)`	Returns the number of contiguous “Cells” that make up the Surface on the specified layer at the Target’s location. Cells are only referenced by the Call `CreatePuddle`, so conversion is recommended for any other use. Try dividing the Cells by 50.5 to get an approximate Radius.
`GetSurfaceTurns([in](GUIDSTRING)_Target, [in](STRING)_SurfaceLayer, [out](INTEGER)_SurfaceTurns)`	Returns the number of Lifetime Rounds remaining for the Surface on the specified layer at the Target’s location.
`GetSurfaceTypeIndex([in](STRING)_Surface, [out](INTEGER)_Index)`	Returns a whole number between 0 and 75 that represents the enumerated value from the list of Surfaces based on the provided Surface’s Name.
`IsInDangerousSurfaceFor([in](REAL)_X, [in](REAL)_Y, [in](REAL)_X, [in](CHARACTER)_Character, [in](REAL)_Radius, [in](INTEGER)_AlsoIfCanStopMovement, [out](INTEGER)_IsDangerous)`	Returns whether the area within a radius centered at the specified coordinates would be dangerous for a specified target (which must be a character). If 1 is passed for `AlsoIfCanStopMovement`, the resulting boolean will take movement into account when determining `IsDangerous`

Query	Notes
`DrawSurfaceOnPath([in](GUIDSTRING)_OwnerObject, [in](GUIDSTRING)_FollowObject, [in](STRING)_SurfaceType, [in](REAL)_Radius, [in](REAL)_LifeTime, [out](INTEGER64)_SurfaceActionHandle)`	SPECIAL QUERY - This works like a Call, telling the entity specified as the FollowObject to create a Surface with the specified radius and lifetime at its location whenever it moves. The Surface will be “owned” by whatever is passed as the OwnerObject. The Query returns a string of numbers and letters necessary to stop the entity from creating the Surface. It is recommended to pass the SurfaceActionHandle to a temporary Database in the THEN portion, to be used by a Call later. See the Trails section under Passives earlier in this guide for examples.

Calls

The following Calls can bring exciting new behaviors to your Surfaces, though there are certainly limitations.

Call	
`CreateSurface((GUIDSTRING)_Target, (STRING)_SurfaceType, (REAL)_Radius, (REAL)_Lifetime, (GUIDSTRING)_Source)`	Requires a Target and optionally a Source, but otherwise behaves the same as CreateSurface does when put as part of a spell’s SpellProperties.
`CreateSurfaceAtPosition((REAL)_x, (REAL)_Y, (REAL)_Z, (STRING)_SurfaceType, (REAL)_Radius, (REAL)_Lifetime, (GUIDSTRING)_Source)`	Works exactly like the above Call, except that it accepts a set of coordinates instead of a Target.
`CreatePuddle((GUIDSTRING)_Target, (STRING)_SurfaceType, (INTEGER)_CellAmountMin, (INTEGER)_CellAmountMax, (INTEGER)_GrowAmountMin, (INTEGER)_GrowAmountMax, (REAL)_GrowTime)`	Creates a puddle of the Surface, in the same manner a creature leaves a randomly shaped pool of blood when injured or killed. The Surface’s final size is random between the CellAmountMin and Max, and will grow in random increments between GrowAmountMin and Max, over the time specified in GrowTime (which can be influenced by SurfaceGrowTimer in the Visual Section). Setting the Min and Max to the same number removes much of the randomness of the puddle, though it will still spread in a random direction each step.
`RemoveSurfaceLayer((GUIDSTRING)_Target, (STRING)_SurfaceLayer, (REAL)_Radius)`	Works like creating a None surface. Requires a Target.
`RemoveSurfaceLayerAtPosition((REAL)_x, (REAL)_Y, (REAL)_Z, (STRING)_SurfaceLayer, (REAL)_Radius)`	Works exactly like the above Call, except that it accepts a set of coordinates instead of a Target.
`TransformSurface((GUIDSTRING)_Target, (STRING)_TransformType, (STRING)_TransformLayer, (REAL)_Radius, (REAL)_Lifetime, (GUIDSTRING)_Owner)`	Works like SurfaceChange, accepting the same changes as TransformType, except only on a singular layer, and limited to a specified radius, and a custom lifetime, and the owner can be set. Requires a Target.
`TransformSurfaceAtPosition((REAL)_X, (REAL)_Y, (REAL)_Z, (STRING)_TransformType, (STRING)_TransformLayer, (REAL)_Radius, (REAL)_Lifetime, (GUIDSTRING)_Owner)`	Works exactly like the above Call, except that it accepts a set of coordinates instead of a Target.
`StopDrawSurfaceOnPath((INTEGER64)_SurfaceActionHandle)`	When passed the SurfaceActionHandle created by a DrawSurfaceOnPath Query, tells the entity creating that Surface to stop creating it. See the Trails section under Passives earlier in this guide for examples.

Anubis/Constellation Code

This guide suffers from a lack of Anubis knowledge (and of lua in general). It seems that the majority of Osiris Queries and Calls are usable in Anubis, but proceed with caution. If useful things are known or discovered, please inform the author of this guide. The following lines of code seem promising.

```
FindSurface(me, 1.0, 3.0, SurfaceType.LAVA)
——————————————
   nodes.AcidTrailSpawn = Proxy{
        game.states.SurfaceTrailSpawner,
        params = {
            SurfaceType = "ACID"
        }
    }
——————————————
IsInSurface(me, SurfaceType.ASH, 0.3)
——————————————
RemoveSurfaceOfTypeAt(SurfaceType.WATER_CLOUD_ELECTRIFIED, trigger1, 30)
```

Other

Unknown modifiers in StatusData, which do not appear to be accessible in the TK:
"AbsorbSurfaceRange","ConstantInt"
"AbsorbSurfaceType","FixedString"

Troubleshooting

I can see my Surface's name when I hover my cursor over where it should be, but it is invisible.

- Check the Visual section and make sure you are using visuals and instances compatible with the layer (Ground or Cloud) that it is meant to be.
- Make sure the Instance or Decal Material have actual graphics, as some may be invisible.
- Zoom out and confirm that the visual is not appearing high above or below the ground.
- If using only Instances to show your Surface, make sure the Minimum and Maximum Surface Radius are correct.

My surface does not appear at all.

- Make sure it's not simply invisible by hovering your cursor over the area you tried to have the Surface appear in. If the Surface's Display Name appears, troubleshoot the visuals.

- Try changing the Surface being created by the spell or effect to one you know that works. If that Surface won't appear, there's likely something wrong with the spell or effect, not the Surface.
- Make sure you are using the correct name for the Surface. The Name field in the Base section should be the same as the original Surface you overrode.
- When adding it to a spell with `CreateSurface`, make sure you did not accidentally include the word "Surface" in the name of the Surface in SpellProperties.
- Check the lifetime in the Behavior section, as well as where it is defined in the spell. If they are both 0, your Surface will disappear immediately upon creation. Change one or both to a higher value and try again. Remember that the Always Use Default Lifetime checkbox, if checked, will override the spell's lifetime.
- Check the size of the Surface in the spell. It should be higher than 0.

My Surface's effects are disappearing too soon or triggering at the wrong time.

- Check the Status Data in the Root Template, and ensure that the following are set correctly:
  - Apply Type
  - Chance
  - Don't Tick While On Surface
  - Duration
  - Force.
  - Keep Alive
  - Only Once Per Turn
- Check the Boost itself and ensure that the following are set correctly:
  - TickType
  - RemoveConditions
  - RemoveEvents
  - TickFunctors

I can't get a Trail to start/stop spawning, or multiple entities are stopping when only one should.

- If you are using a Passive, make sure the Passive is applying a Boost. The Boost doesn't need to have anything in its fields, as Osiris just needs an Event or Query it can use to trigger the rest of the code.
- If you can't get the Surface to start, try temporarily adding the Call `DebugText(_Entity, _String)` with the same entity as its target and some identifying text such as "TRAIL START" as the String. This String will appear above the entity's head briefly when the conditions are met.
  - If the String appears but the trail does not begin:
    - Look for any typos in the Surface name in the Osiris code.
    - Check that the Surface itself works when summoned other ways.
    - Ensure that the `DrawSurfaceOnPath` Query is being used as a Query (just after an AND) and not as a Call (after the THEN).
  - If the String does NOT appear, the conditions are not being properly met, and should be adjusted.
- If you can't get the Surface to stop, it is likely that it is not receiving the SurfaceHandle. Make sure that the Handle is being saved in a Database in the Calls of the starting-trail code. If multiple entities are creating trails at the same time, check whether the wrong entity is being told to stop drawing the trail.
- If all the entities stop simultaneously when you want them to stop individually:

- The Database may only have the SurfaceHandle and no way to associate them with a specific entity. Include the entity in the Database entry you create.
- You may have set the SurfaceHandle instead of letting it generate one automatically. If so, you will need to let it generate or include code to differentiate the SurfaceHandles belonging to each entity.
- You may have forgotten to remove Database entries that were previously stopped, which can be especially noticeable when the SurfaceHandle is set instead of generated, or if you wish for the entity to draw a different Surface and then attempt to switch it back to the first Surface.